UC-404

SAND95-8223
Unlimited Release
Printed April 1995

Optimal Heat Transfer Design of Chemical Vapor Deposition Reactors

Christopher D. Moen, Paul A. Spence
Computational Mechanics Department
and
Juan C. Meza
Scientific Computing Department
Sandia National Laboratories
Livermore, CA 94551-0969

ABSTRACT

A new tool for optimal heat transfer design has been constructed by coupling the OPT++ optimization library to the TACO2D finite element heat transfer code. The optimization heat transfer code can be used to quickly and efficiently find optimal operating parameters required for target design criteria. This tool has been applied to the heat transfer design of a rotating disk chemical vapor deposition reactor. The results from two reactor design problems indicate that optimal solutions can be found quickly and efficiently.

1 Introduction

Computational simulation is a powerful tool for the design of parts and equipment for material processing. Physically based numerical models can provide insight on heat transfer, fluid mechanics, chemical kinetics, and structural response for a given design concept. The integration of computational analysis into the design process can reduce development time and costs while improving product performance. Computer models allow designs to be prototyped in software thus eliminating the need for the slow and expensive cycle of "build and test". The typical goal for a design process is to achieve the best possible performance within a given set of constraints. Trial-and-error is the common approach used to address this design optimization problem. The drawback with the trial-and-error evaluation is that it is time consuming (both with software and hardware) and it does not guarantee that the best design is found.

We have developed a capability for addressing design optimization problems through linking optimization software to large-scale analysis codes. This software tool allows a designer to quickly determine optimal design parameters using computational simulation. This report describes the software tool and the application of this software capability to the optimal design of rotating disk chemical vapor deposition (CVD) reactors. The software tool is constructed from existing software packages developed at Sandia: the OPT++ optimization library [1] is linked to the TACO2D finite-element thermal analysis code [2].

The OPT++ optimization software is an object-oriented library of optimization routines. Object-oriented programming allows for greater portability and applicability to different applications. One of the main features of this approach is that little knowledge of object-oriented programming is required by the end user in order to use the OPT++ software. In practice, the end user need only supply an interface routine that calls the TACO2D heat transfer code. This interface routine is called by OPT++ to provide the response to variations in the design parameters.

The TACO2D finite-element heat transfer code was chosen for the optimization tool because it has several features that are used extensively in the design of reactor systems. The code has variable thermophysical properties, multiple enclosure radiation, contact resistances, and a large set of boundary conditions. The enclosure radiation feature, combined with partially transmitting windows, makes the governing equations highly nonlinear and difficult to solve. It is also the code we are most familiar with, accelerating the integration process with the optimization software.

In previous heat transfer design approaches, the optimization process involved examining the local behavior of the temperature field as a function of the design parameters.

This approach involved running the TACO2D code manually for different combinations of the design parameters. Because the optimization problem is nonlinear, the optimal solution was difficult to find and several iterations on the local map construction were required. The use of the OPT++ nonlinear optimization software makes the optimization process less labor-intensive and more computationally efficient. At present, the optimization is limited to parameter optimization for a fixed geometry. In the future, we will address geometry optimization which requires more efficient grid generation and view factor computations. The candidate heat transfer code for these more complicated optimization problems is the COYOTE2 code [3, 4]. This code has the desirable features of faster view factor calculations and moving meshes.

The report is organized to first give and overview of how the software tool is constructed and then give examples of how the code is used to optimize the design of two different rotating disk reactors. The various optimization methods available within the OPT++ software package are discussed in Section 2. The concept of object-oriented classes for optimization methods is introduced. The interface between the optimization code and the analysis code is code-dependent so the mechanics of linking the TACO2D heat transfer to the OPT++ optimization library are discussed in Section 3. This information can be used as a model to link any large-scale simulation code to the optimization library. Two sample heat transfer optimization applications are presented in Section 4. Results of the heat transfer design optimization are presented and the various optimization methods are discussed.

2 Object-Oriented Optimization Methods

The OPT++ software consists of a library of object-oriented optimization algorithms written in C++. Unlike procedural programming which emphasizes the development of algorithms to accomplish a specific task, object-oriented programming relies on the implementation of new data types. The major difference in object-oriented programming is the ability to create user-defined data types and add them to an existing language. It is these new objects that give object-oriented programming its name. Through these new objects a computer language can be easily extended to handle new applications.

One of the major concepts of object-oriented programming is that of a class. A *class* is a user-defined data type that allows the user to concentrate on the use of the new data type by hiding the actual implementation details. A class typically consists of both a data structure and a group of subroutines that can manipulate these data structures. The data inside the structure is hidden from the user in that the only way to access it is through the subroutines defined as part of the class.

Good examples of these ideas are matrix classes for linear algebra. With a matrix class, a user can define vectors and matrices as part of the computer language as well as use the standard matrix operations defined for these objects, such as matrix addition, matrix multiplication, and inversion. For a more complete description of object-oriented programming see [1, 5, 6, 7].

2.1 Nonlinear Problem Classes

There are two main features of the problem classes contained in OPT++. The first feature is the definition of the nonlinear problem. The second feature is the implementation of the objective function.

A general unconstrained optimization problem can be stated as follows:

$$\min_{x \in R^n} f(x).$$

For this problem, the cost or objective function $f(x)$ is assumed to be a general nonlinear function. There are many ways of classifying nonlinear programming problems. OPT++ defines nonlinear programming problems by the availability of analytic derivatives of the objective function. The main advantage of this classification is that users can easily decide what type of problem they have based on whether analytic derivatives are available or not. For example, OPT++ uses the classification displayed in Table 1.

OPT++ also provides 4 classes derived from these 3 base classes. The first 3 classes are called NLF0, NLF1, and NLF2 and have a pre-defined calling sequence. These classes

Table 1: Nonlinear Problem Classification used by OPT++

NLP0	–	No derivative information available
NLP1	–	Analytic first derivatives available
NLP2	–	Analytic first and second derivatives available

can be used to solve some simple optimization problems or can be used as templates for more sophisticated objective functions. The fourth class is called FDNLF1 and is identical to NLF1 except that derivatives are internally computed using finite-differences.

In the TACO2D application, the objective function is computed by using the results of an analysis code that does not provide derivatives. Therefore, the normal classification scheme would declare this problem as an NLP0 object. However, since we would like to use gradient-based methods we chose to declare an FDNLF1 object and have OPT++ compute the derivatives using finite-differences.

The second feature that OPT++ provides is the generality with which the objective function can be implemented. In OPT++, the functions that evaluate the objective function, gradient, and Hessian are defined as virtual functions. This allows the software to defer the definition of how the function, gradient, and Hessian are actually implemented so that users can create their own definitions. The base classes can be thought of as place-holders for the codes that will be called to compute the objective function.

2.2 Optimization Method Classes

There are many classifications possible for optimization algorithms, but most well-known methods can be grouped into one of three classes:

- Direct Search methods
- Conjugate gradient like methods
- Newton-like methods

For example, the Nelder-Mead simplex method falls into the direct search class, the nonlinear conjugate gradient method falls into the conjugate gradient class, and the Newton-like class includes methods such as the quasi-Newton methods. OPT++ contains C++ classes for 5 different methods: 1) a Newton method, 2) a finite-difference Newton method, 3) a quasi-Newton method, 4) a nonlinear conjugate gradient method and 5) a parallel direct search method.

The Newton method is potentially the most powerful method, but it requires the computation of a Hessian matrix which is not available in the TACO2D application. A

finite-difference Newton method is identical to the Newton method but uses finite-differences to compute the Hessian matrix. In practice this option can be as effective as the Newton method, but it can be computationally expensive. The quasi-Newton method is the most attractive in that theoretically it can be nearly as effective as the full Newton methods, but computationally less expensive. Most of our results are based on this option. The nonlinear conjugate gradient method is also attractive in that it is computationally inexpensive, but this method can be slow to converge. Although we tried this method on one sample problem it did not appear effective for this class of problems. The final option, the parallel direct search method, was not used for these problems. For further details on these algorithms see references [8, 9].

3 Linking the Optimizer to the TACO Code

One of the main advantages of object-oriented programming is the relative ease with which new applications can be developed by interfacing existing analysis codes with the optimization methods contained in OPT++. Typically, if the optimization algorithm requires a particular calling sequence the user is forced into writing a subroutine that will interface between the optimizer and the function evaluator. Once the interface is in place, OPT++ minimizes the amount of work that is required to add new optimization methods.

In this section, the mechanics of interfacing the TACO2D heat transfer code to the OPT++ optimization library are discussed. The approach taken in linking the two pieces of software is to be as nonintrusive as possible. A custom built interface is constructed so that the two software packages can communicate. This discussion is intended as a guide for making changes to the problem-dependent interface and also as an example of interfacing any analysis code to the OPT++ libraries.

3.1 TACO2D—OPT++ Interface

Interface routines are required to pass information between TACO2D and OPT++. The TACO2D code, by itself, is a stand-alone heat transfer code. The OPT++ code is a set of object-oriented software libraries containing optimization tools. The coupling between codes is complicated by the fact that the OPT++ libraries are written in C++ while the TACO2D code is written in FORTRAN. All these pieces of code are combined into one executable program. The compiling and linking of these subroutines of different languages is discussed in the Appendix.

The optimization heat transfer code is constructed from four different blocks of code. The code hierarchy is shown in Figure 1. Two existing pieces of code are the analysis code and optimization library. Two pieces of code must be written by the user: the top-level program and the interface code. The top-level program (written in C++) manipulates the optimization tools. It makes calls to the analysis code through an interface subroutine. The interface code extracts information from the analysis code and performs other problem-dependent tasks. For the TACO2D interface, there are two levels of interface code to facilitate the communication of information between OPT++ and TACO2D. The top-level interface code (written in C) is used to manipulate input and restart files for the analysis code, read the target parameters, and compute the objective function. The objective function is a measure of how close the solution is to the target design criteria and is generated by running the analysis code for a particular set of input parameters. In order to extract the proper information from the analysis code, a second-level interface, called TACO2DOPT,

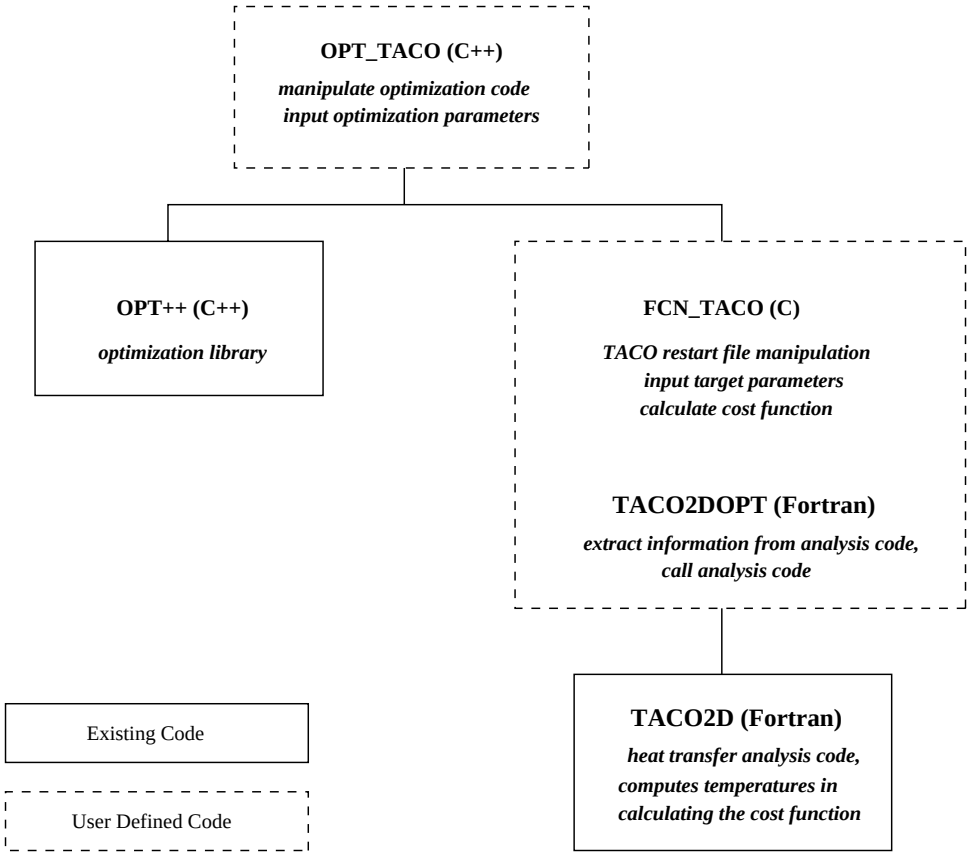

Figure 1: Code hierarchy for TACO2D—OPT++

is required. The second-level interface (written in FORTRAN) calls TACO2D directly and extracts information from COMMON blocks. The only modification to the TACO2D code consists of changing the main FORTRAN program to a subroutine.

The second-level interface code extracts temperatures from the analysis code COMMON blocks for use in the calculation of the objective function. Passing parameters such as heating rates or boundary conditions into the analysis code can sometimes be tricky since they are usually passed into the analysis code through an input file. Writing filters for input decks can be tedious, but fortunately the TACO2D code has a feature that simplifies passing those parameters—user-defined subroutines. In our case, the optimization parameters are passed into the analysis code through user-defined subroutines (page 28 of the TACO manual [2]) and the interface code has direct access to those subroutine COMMON blocks.

3.2 Practical Implementation

The heat transfer optimization tool is only useful if it can return solutions in a timely manner. The optimizer calls the analysis code many times in order to find the optimal solution. The analysis code, in turn, must iterate in time to steady-state each time it is called. The time marching method is the most stable and predictable solution strategy for nonlinear problems in the TACO2D code. The duration of the time iteration process is dependent upon how close the initial solution guess is to the steady-state solution. Therefore, it is crucial to begin each cycle through the optimizer with the most current solution reflecting changes in the optimization parameters. Since each call to the TACO2D code acts as a self-contained run, converged solutions must be passed along as restart files. New solutions are passed along by copying the analysis code restart file from the last optimization step to the analysis code input file for the next optimization step. The code copying steps are executed by the interface code through UNIX operating system calls.

Another factor affecting the time to convergence within each call to the analysis code is the definition of convergence. The analysis code must return a steady-state solution to avoid introducing artificial time gradients into the optimizer. The original TACO2D code allows no method of monitoring convergence. It only integrates out to a given fixed point in time. In order to ensure convergence to steady-state, this time must be made large. It was found that a large amount of computational time was unnecessarily used because, with a good initial guess, the time to steady-state was much smaller. A new feature was added to the TACO2D code to monitor convergence when integrating in the unsteady, nonlinear mode. The L_2-norm of the right-hand side of the linearized system (similar to a physical conservation law in a finite volume scheme) is used to determine convergence.

$$||K\left(T^n\right)T^n - F\left(T^n\right)||_2$$

The stiffness matrix is K and the loads are contained in F. When this norm goes to zero, the discrete governing equations are satisfied. This criterion is used because it does not scale with the time step. The only drawback in using a convergence tolerance is in determining how small the L_2-norm must be in order to guarantee convergence to steady-state. This norm scales with the physical size of the problem and the tolerance must be determined *a priori*.

4 Heat Transfer Design Optimization

Two test cases are presented to demonstrate the design optimization capabilities of the OPT/TACO tool. Both cases involve design aspects of rotating disk reactors. Rotating disk chemical vapor deposition reactors are commonly used for growing opto-electronic devices. A critical factor in this process is the temperature distribution on the substrate material which directly affects the quality of the final electronic device. A common time consuming task in the design analysis of these reactors is to define optimal heater powers and their placement in order to achieve a prescribed temperature profile on the wafer surface.

The first case is a design for a model problem configuration of a rotating disk CVD reactor. The heater power densities are optimized to give a desired wafer temperature profile. The results of the optimization indicate an interesting relationship between the heating elements that may prove useful in optimal control. The second case is more representative of an actual rotating disk reactor design. In this problem, the contact resistance inside the wafer carrier is optimized to give a desired wafer temperature. The optimization of contact resistance indicates that a wafer carrier of variable thickness may improve temperature uniformity.

The numerical issues associated with the optimization are discussed in each case. These "lessons learned" provide useful information on the efficient use of the optimization heat transfer design tool.

4.1 Power Density Optimization

This model problem is created to demonstrate heater power density optimization. The optimization study is used to explore power density requirements for different wafer temperature settings. The configuration is a simplified model of a rotating disk reactor consisting of a wafer carrier, four radiative heaters, and a heat shield. The rotating drive shaft and support assembly have been removed. The carrier/heater assembly sits inside a stainless steel reaction chamber with cooled walls. The reactor is axisymmetric so only a two-dimensional cross-section is modeled. The discrete wafers are modeled as a continuous annular ring on the carrier. A close-up view of the wafer carrier, heaters, and heat shield is shown in Figure 2. The wafer carrier rotates about the vertical axis. The cross-section of the reactor configuration including the reaction chamber is shown in Figure 3. The graphite wafer carrier has a radius of 2.5 inches and a thickness of 0.3 inches. A sapphire wafer 25 mils thick and 2.0 inches in diameter is embedded in the top surface of the wafer carrier, 0.25 inches away from the center. Four graphite heating elements are placed beneath the wafer carrier. A molybdenum heat shield is placed beneath the heating elements to reflect heat back towards

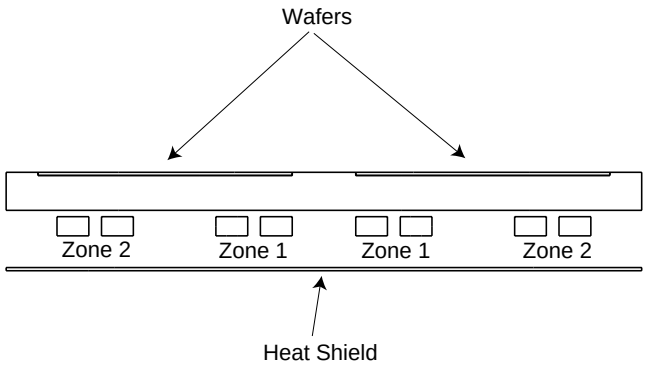

Figure 2: Enlarged View of Model Problem Carrier/Wafer Configuration with Heat Shield

the wafer carrier.

The heating is introduced in the finite-element model using uniform volumetric heating within the heating elements. The heating elements radiate and form a radiation enclosure with the heat shield, wafer carrier, and the reaction chamber. The radiation surfaces are modeled as diffuse gray surfaces. Each material is assumed to have a constant emissivity, shown in Table 2. The wafer and carrier surfaces are cooled by convection and enclosure radiation to the cooled reactor chamber walls. A convection coefficient of 500 $W/m^2/K$ is applied to the outside of the canister with a reference temperature of 300 K. The outside of the canister also radiates to a uniform background of 300 K with an emissivity of 0.3. The convection coefficient applied to the wafer and to the top side of the wafer carrier is 30 $W/m^2/K$ with a reference temperature of 300 K. There is a contact resistance of 8000 W/m/K applied between the wafer and the wafer carrier.

The optimization target is to have a uniform temperature across the surface of the wafer, achieved by varying the heating element power densities. The power is delivered to the heating elements such that certain groups of heaters have the same power density. These groups are called zones and each zone is on a separate power circuit. The power is varied in two zones, making this a two parameter optimization. The two inboard heaters are Zone 1 and the two outboard heaters are Zone 2.

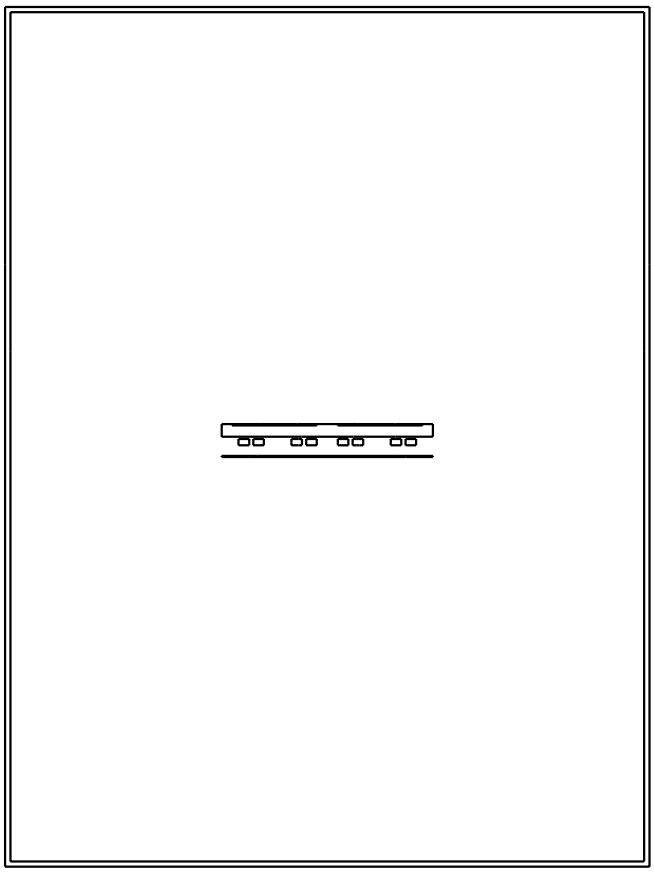

Figure 3: Model Problem Rotating Disk Reactor with Enclosure

Table 2: Material Surface Emissivities for Model Problem

Material	Emissivity
Stainless Steel	0.3
Molybdenum	0.4
Graphite Heater	0.95
Graphite Carrier	0.95
Sapphire Wafer	0.9

The wafer temperature and uniformity in the optimization process are found by minimizing the objective function with respect to the power densities. The objective function is a function of power density, though it is calculated from the wafer surface temperatures. The objective function is constructed from the difference between the discrete nodal temperatures, T_i, on the wafer surface and the target temperature,

$$f = \left[\sum_i (T_i - T_{target}) \right]^{\frac{1}{2}}.$$

Note that this is an unconstrained optimization and no restrictions are placed on allowable temperatures within the heating elements.

The heater power densities are optimized for three different wafer temperature settings. These may be three different processing temperatures that occur along a process path. The optimum zone power densities for the three different target temperatures are shown in Table 3. The temperature variation across the wafer increases with increasing wafer temperature and heater power. The corresponding temperature distributions are shown in Figure 4. An increase in uniformity by variations in heater configuration can only be achieved by changing the heater placement or breaking up the two zones into three or four zones.

Table 3: Optimized Heater Power Densities for Model Problem Reactor

Target	Zone 1	Zone 2	Temperature Variation
1000 K	28.2 MW/m^3	117. MW/m^3	$\pm$8 K
1125 K	42.3 MW/m^3	178. MW/m^3	$\pm$13 K
1250 K	63.3 MW/m^3	261. MW/m^3	$\pm$20 K

The different optimization points may be set points for a larger process. The surface temperature may be ramped up and down within the processing steps. By plotting the temperature set points as a function of the two optimal zone heater power densities, it is found that there is a linear relation between power densities, shown in Figure 5. This is an interesting observation from the point of optimal control. This figure indicates that the power densities for optimum wafer temperature can be controlled by a single parameter. The two zonal power densities always have a fixed ratio. This path between different wafer temperatures may not be optimal in terms of shortest time to change the temperature, but the figure provides a quick way of determining the end point power densities.

There have been several lessons learned about successfully running the OPT/TACO heat transfer design tool. These lessons are outlined below in context to the rotating disk model problem. The first lesson is that round-off errors can cause the optimizer to fail if

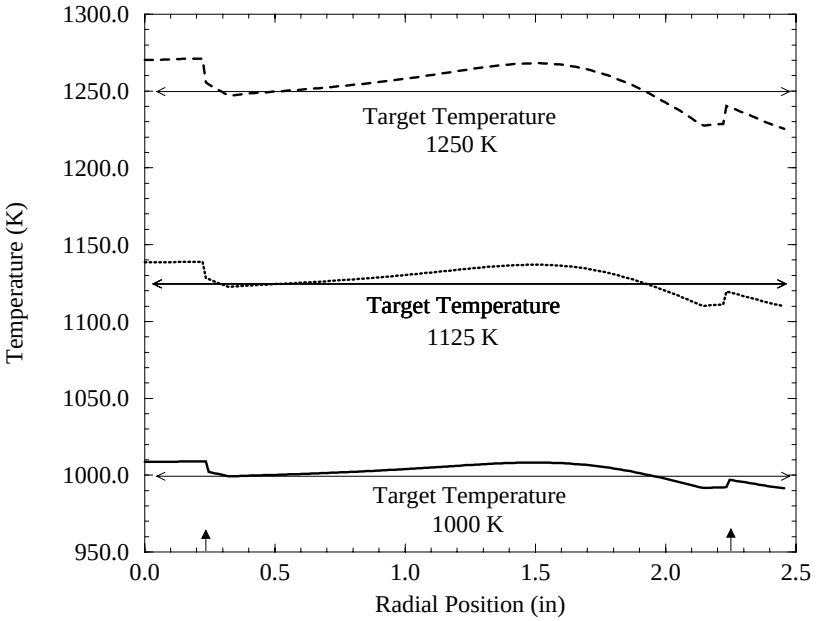

Figure 4: Optimized Surface Temperatures for Model Problem Reactor

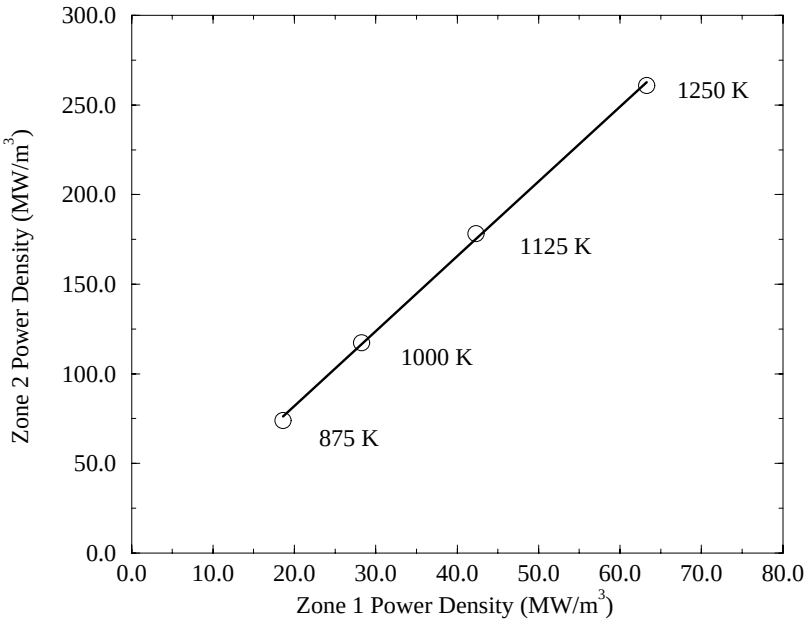

Figure 5: Power Densities for Given Target Temperatures

scaling is not properly addressed. The optimization parameters should be scaled to order one when passed to the optimization routines. These numbers are scaled back to their physical dimensions for use in the analysis code. If the parameters are too large when they are passed into the optimizer, the associated numerical round-off errors adversely affect the convergence of the optimization routines. Consider the example where the heater powers for the CVD reactor model are optimized to give a desired wafer temperature distribution. The heater power densities are expected to be on the order of 1×10^7 W/m^3. The initial guess at heater power for the optimization routine is set to 1 and this number is scaled by 10^7 for use in the analysis code. The optimizer works with numbers of order one while the heat transfer code works with the actual physical size.

For most optimization problems, the use of forward-differencing to numerically evaluate function gradients is sufficient for convergence. The solutions to the power density design problems were found using a quasi-Newton method. Two different methods for calculating objective function gradients were tested: central-differences and forward-differences. Although both methods converged in the same number of iterations, the forward-differencing took less time to reach the optimal solution than the central-differencing. The forward-differenced solution required 12 minutes of CPU time on a HP 735 workstation, running in double precision arithmetic. The central-differenced solution required 20 minutes of CPU time. For most problems, the forward-differencing is accurate enough to find the optimum solution. In some cases the central-differencing adds enough accuracy to converge to a better solution than the forward-differencing. It was hypothesized that the increased accuracy of the central-differencing might increase the convergence rate. Instead the convergence rate is the same as the forward-differencing at twice the cost. The solution strategy is to use the forward-differencing to converge to near the solution as fast as possible. Then the central-differencing is used to refine the solution.

The time required to find the optimum solution also depends upon how good the initial guess to the solution is. Knowledge of the behavior of the objective function as a function of optimization parameters can be used to find a good initial guess. A contour map of the objective function with respect to the optimization parameters provides such information. A contour map of the zonal power densities for the target temperature of 1250 K is shown in Figure 6. In this map, the optimal solution is in the middle of the broad flat central valley. The valley is bounded by steep sides. The optimizer quickly finds its way to the valley floor. It then spends most of the time bouncing along the sides of the valley on its way to the minimum. The purpose of the nonlinear optimizer is to find the minimum point in fewer function evaluations (calls to the heat transfer code) than are required to generate a function map and read the minimum point manually.

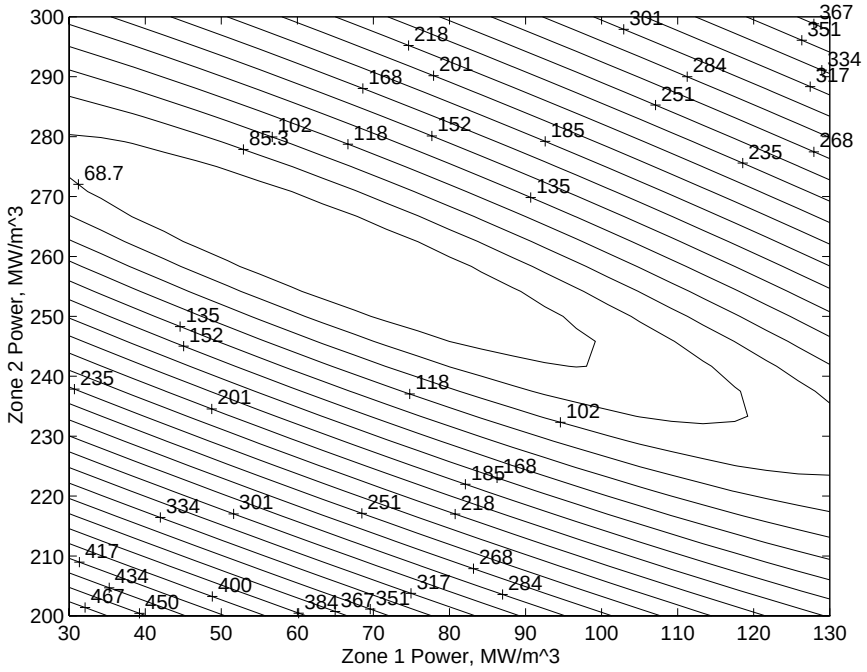

Figure 6: Function Map for Generic Reactor, T=1250 K

The optimization methods available in the OPT++ library are all unconstrained. It is possible for the optimizer to select a set of parameters that are physically meaningless to the designer. The optimal design solutions for the rotating disk reactor do not fall across any "would be" constraint boundaries, but the optimizer sometimes overshoots the boundary of physically realizable solutions during the transient steps. This is a critical issue for the model problem because negative power densities are sometimes selected. In the early phase of the optimization process, the step size became large causing the optimizer to attempt to evaluate the function at points that correspond to negative heater powers. Since this is physically unreasonable, the heat transfer code fails. The problem is solved by placing an upper bound on the allowable step size in the optimization method. The maximum step size is problem dependent and is determined empirically. In the future, the OPT++ library will contain constrained optimization methods which will allow for a better treatment of these problems.

4.2 Contact Resistance Optimization

This test problem is presented to demonstrate the use of radial variation in wafer carrier assembly thickness to achieve uniform wafer temperature. The radial thickness parameter is modeled with a contact resistance and the thickness profile can be backed out of the contact resistance profile. The use of the internal boundary condition to model changes in geometry

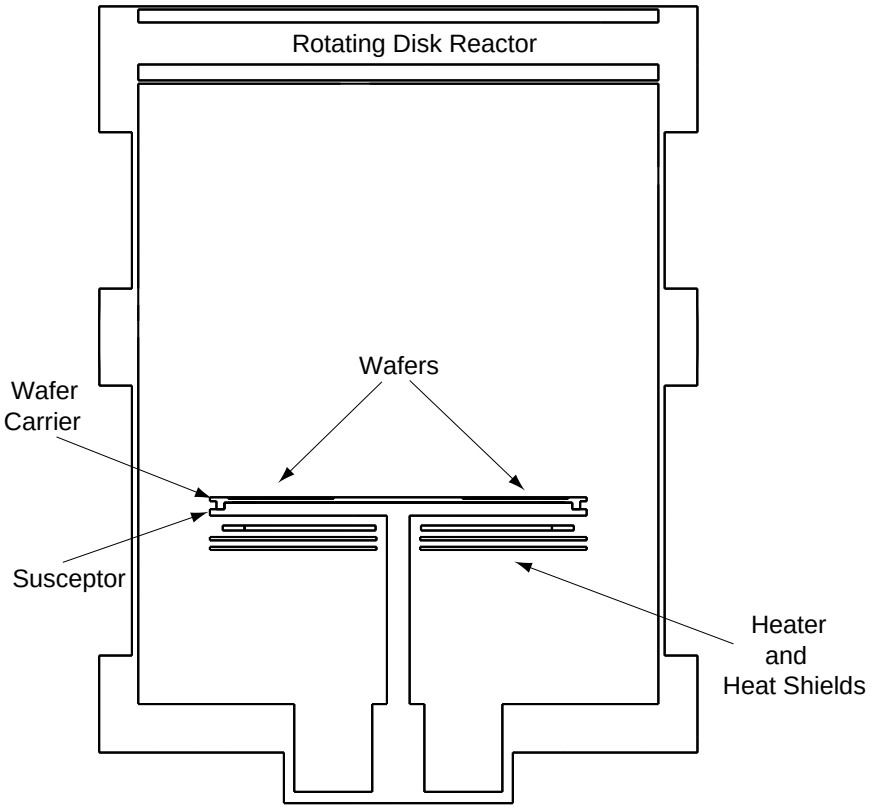

Figure 7: Rotating Disk Reactor Configuration

avoids remeshing as the geometry changes.

The geometry used for this model is a more complete description of a rotating disk reactor than the model problem discussed previously. In this respect, it is a more difficult problem to solve. The finite-element modeling is carried out in a manner similar to that of the model problem. Only a cross-section of the axisymmetric reactor is modeled and the discrete wafers on the wafer carrier are modeled as a continuous ring. The cross-section of the reactor configuration is shown in Figure 7. In this configuration, the wafers are attached to an assembly which consists of a carrier and a susceptor. The wafer rests on the wafer carrier which sits on top of the susceptor. The wafer carrier and susceptor assembly spins and is heated from below.

The wafer temperature is refined to a target temperature by changing the contact resistance between the wafer carrier and susceptor. The contact resistance is a modeling parameter that represents the thermal resistance caused by the small gap between the wafer carrier and susceptor. This provides for radial control of heat transfer to the wafer and refinement of the wafer temperature uniformity. For modeling purposes, the gap variation is broken up into four zones. The target temperature is 1175 K and it is assumed that heating

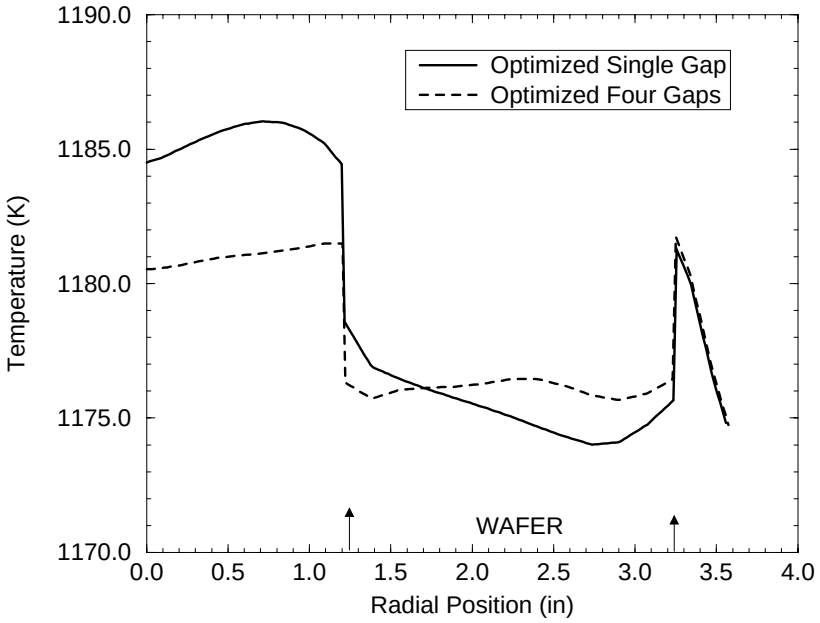

Figure 8: Wafer Surface Temperature

power is fixed.

A baseline solution is established by finding the optimal uniform gap spacing required to achieve the target temperature for a prescribed heater power density. The four zone optimization problem is then run for the same heater power density. Varying the contact resistance allows the wafer surface temperature tolerance to be tightened. The wafer temperatures before and after the four parameter optimization are shown in Figure 8. The baseline solution exhibits a 4 K variation in temperature, but the optimized solution is refined to a 0.5 K variation. The mean temperatures are above the target temperature for both cases. This is thought to be a function of the temperature jump at the end points of the wafer. The optimized contact resistance leads to a wafer carrier/susceptor interface with variable gap spacing, shown in Figure 9. The gap spacing is exaggerated for the purpose of visualization.

Several lessons were learned in trying to solve this larger optimization problem. This problem is more difficult to solve than the model problem because it has more unknowns, it is more nonlinear, and there are more optimization parameters. The first lesson involves numerical round-off errors. These optimization problems have been run with 32-bit and 64-bit arithmetic. In all cases, the optimization routines were run with 64-bit arithmetic. Only the precision in the TACO2D code varied. Numerical round-off errors from the 32-bit "single precision" arithmetic cause the optimization routines to stop prematurely. Although the actual temperatures calculated by the heat transfer code in 32-bit and 64-bit mode are

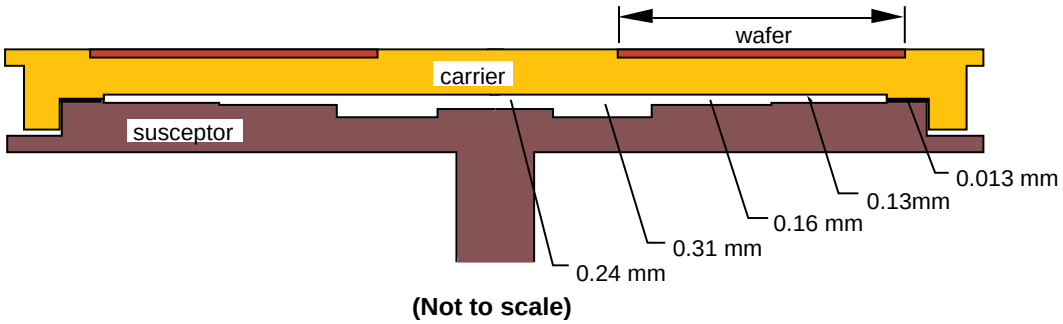

Figure 9: Carrier/Susceptor Gap Spacing for Optimized Contact Resistance

equal by engineering standards, the loss of accuracy is enough to cause convergence problems within the optimizer.

The quasi-Newton optimization method shows the best performance for these rotating disk design problems. Various optimization algorithms available in the OPT++ library were compared to determine the most efficient way of solving these design problems. For the "contact resistance problem", the quasi-Newton and finite-difference Newton methods and a nonlinear conjugate gradient method were tried. In addition, for the quasi-Newton method, both central and forward-differences for the gradient evaluations were tried. The results are shown in Table 4. The nonlinear conjugate gradient was the worst method of all of the methods tried. All three of the Newton methods converged in essentially the same number of iterations. The major difference between the methods is in the computation of derivatives. For this particular problem, the best approach was to use forward-differences for the computation of the first derivatives and a quasi-Newton approximation to the second derivatives.

Table 4: Optimization Method Performance for Contact Resistance Problem

Optimization Method	Objective Function value	Iter	Function Calls	Time(minutes)
Nonlinear CG	1.1283	100	1940	4162
Finite-Difference Newton	1.0094	22	428	926
Quasi-Newton (FD)	1.0094	20	120	251
Quasi-Newton (CD)	1.0094	20	204	432

5 Summary

A new tool for optimal heat transfer design has been presented. The design tool is constructed from existing pieces of software: the OPT++ optimization library and the TACO2D finite element heat transfer code. The code integration process is simple and nonintrusize because of the object-oriented construction of the OPT++ library. The code integration procedure outlined in this report can be generalized to a procedure for adding optimization capabilities to any large-scale simulation code.

The OPT/TACO software tool allows for the rapid prototyping of reactor designs. This design tool has been applied to the heat transfer design of rotating disk chemical vapor deposition reactors. Results for two reactor design problems are presented along with practical strategies for performing the optimization. The optimization design tool, as it exists now, can be used for parameter optimization. Future work for heat transfer optimization involves adding constraint features and looking towards geometric optimization.

References

[1] J. C. Meza. "OPT++: An Objet-Oriented Class Library for Nonlinear Optimization". Report SAND94–8225, Sandia National Laboratories, Livermore, CA, March 1994.

[2] W. E. Mason. "TACO3D—A Three-Dimensional Finite Element Heat Transfer Code". Report SAND83–8212, Sandia National Laboratories, Livermore, CA, April 1983.

[3] D. K. Gartling and R. E. Hogan. "Coyote II—A Finite Element Computer Program for Nonlinear Heat Conduction Problems, Part I—Theoretical Background". Report SAND94–1173, Sandia National Laboratories, Albuquerque, NM, October 1994.

[4] D. K. Gartling and R. E. Hogan. "Coyote II—A Finite Element Computer Program for Nonlinear Heat Conduction Problems, Part II—Users Manual". Report SAND94–1179, Sandia National Laboratories, Albuquerque, NM, October 1994.

[5] Timothy Budd. *An Introduction to Object-Oriented Programming.* Addison-Wesley, Reading, MA, 1991.

[6] Allen I. Holub. *C+ C++ Programming With Objects in C and C++.* McGraw-Hill, New York, NY, 1992.

[7] B. Stroustrup. *The C++ Programming Language.* Addison-Wesley, Reading, Massachusetts, 1987.

[8] Jorge J. More and Stephen J. Wright. *Optimization Software Guide.* SIAM Press, Philadelphia, PA, 1993.

[9] J.E. Dennis and R.B. Schnabel. *Numerical Methods for Unconstrained Optimization and Nonlinear Equations.* Prentice Hall, Englewood Cliffs, New Jersey, 1983.

A Compiling and Linking

The mixture of C++, C, and FORTRAN languages required to construct the optimization heat transfer code may seem intimidating, but is actually straightforward. The code has been constructed on a SGI/R8000, a SGI/R4000, and a HP 735/125. On the HP 735, the native FORTRAN and C compilers were used, but the C++ compiler was the GNU g++ compiler. (This is available through a no-fee license from the Free Software Foundation and can be downloaded through the Internet from the anonymous ftp site: gatekeeper.dec.com, /pub/GNU.)

Each type of code is compiled to the object code level (.o) with the specific compiler. Then all the object code is linked together with the C++ compiler. The FORTRAN system libraries must be included in the call to the linker, for example:

$$\mathrm{C++} \quad -o\ \mathrm{opt\_taco} \quad \$(\mathrm{OBJ\_CODE}) \quad -l\mathrm{cl} \quad -l\mathrm{U77} \quad -l\mathrm{m}.$$